Hit List

First Hit | Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 3 of 3 returned.

☐ 1. Document ID: US 20020143795 A1

Using default format because multiple data bases are involved.

L1: Entry 1 of 3 File: PGPB Oct 3, 2002

PGPUB-DOCUMENT-NUMBER: 20020143795
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020143795 A1

TITLE: Improved method and system of computer file management

PUBLICATION-DATE: October 3, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Fletcher, Thomas O.P.	Ottawa		CA
Van Der Veen, Peter H.	Kanata		CA
Dodge, Dan	Ottawa		CA

US-CL-CURRENT: 707/200

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

☐ 2. Document ID: US 6385625 B1

L1: Entry 2 of 3 File: USPT May 7, 2002

US-PAT-NO: 6385625
DOCUMENT-IDENTIFIER: US 6385625 B1

TITLE: Highly available cluster coherent filesystem

DATE-ISSUED: May 7, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Slaughter; Gregory L.	Palo Alto	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Palo Alto	CA			02

APPL-NO: 09/528537 [PALM]
DATE FILED: March 20, 2000

PARENT-CASE:
This appln. is a continuation of Ser. No. 09/069,013 filed Apr. 28, 1998, now U.S. Pat. No. 6,058,400.

INT-CL: [07] G06 F 17/30

US-CL-ISSUED: 707/201; 707/200, 707/202
US-CL-CURRENT: 707/201; 707/200, 707/202

FIELD-OF-SEARCH: 707/200, 707/201, 707/202, 707/10, 707/100, 707/203, 707/204, 707/205, 707/103

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5890161	March 1999	Helland et al.	707/103
5892900	April 1999	Ginter	713/200
5916307	June 1999	Piskiel	709/300
5960200	September 1999	Eager	395/705
6249792	June 2001	Zwilling et al.	707/205

ART-UNIT: 2771

PRIMARY-EXAMINER: Mizrahi; Diane D.

ATTY-AGENT-FIRM: Conley, Rose & Tayon, PC Kivlin; B. Noel

ABSTRACT:

A distributed filesystem operating on a distributed computing system uses existing local filesystems as building blocks. A cluster filesystem layer is added above the local filesystems to handle functions necessary for a distributed filesystem, such as data coherency. A meta-data stub is developed to flush meta-data from the local filesystem to a storage device. In one embodiment, the meta-data stub is a copy of the flush portion of the meta-data portion of a local filesystem. Additionally, the distributed filesystem supports a plurality of coherency algorithms and variable granularity. A user may select the coherency algorithm that best suits the computer system and granularity that best suits the access patterns of the computer system.

20 Claims, 5 Drawing figures

☐ 3. Document ID: US 6058400 A

L1: Entry 3 of 3 File: USPT May 2, 2000

US-PAT-NO: 6058400
DOCUMENT-IDENTIFIER: US 6058400 A

TITLE: Highly available cluster coherent filesystem

DATE-ISSUED: May 2, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Slaughter; Gregory L.	Palo Alto	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Palo Alto	CA			02

APPL-NO: 09/069013 [PALM]
DATE FILED: April 28, 1998

INT-CL: [07] G06 F 17/30

US-CL-ISSUED: 707/201; 707/200, 707/202, 707/1, 707/2, 707/3, 707/4, 707/5, 707/10, 707/100
US-CL-CURRENT: 707/201; 707/1, 707/10, 707/100, 707/2, 707/200, 707/202, 707/3, 707/4, 707/5

FIELD-OF-SEARCH: 707/1, 707/2, 707/3, 707/4, 707/5, 707/10, 707/201, 707/200, 707/202, 707/100, 709/300, 395/701, 395/705, 395/706, 713/200

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5892900	April 1999	Ginter	713/200
5916307	June 1999	Piskiel	709/300
5960200	September 1999	Eager	395/705

ART-UNIT: 271

PRIMARY-EXAMINER: Black; Thomas G.

ASSISTANT-EXAMINER: Mizrahi; Diane D.

ATTY-AGENT-FIRM: Conley, Rose & Tayon, PC Kivlin; B. Noel

ABSTRACT:

A distributed filesystem operating on a distributed computing system uses existing local filesystems as building blocks. A cluster filesystem layer is added above the

local filesystems to handle functions necessary for a distributed filesystem, such as data coherency. A meta-data stub is developed to flush meta-data from the local filesystem to a storage device. In one embodiment, the meta-data stub is a copy of the flush portion of the meta-data portion of a local filesystem. Additionally, the distributed filesystem supports a plurality of coherency algorithms and variable granularity. A user may select the coherency algorithm that best suits the computer system and granularity that best suits the access patterns of the computer system.

25 Claims, 5 Drawing figures

Term	Documents
CUSTOM	119089
CUSTOMS	2637
FILESYSTEM	1222
FILESYSTEMS	341
(FILESYSTEM NEAR CUSTOM).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	3
(CUSTOM NEAR FILESYSTEM).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	3

Hit List

First Hit | Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 9 of 9 returned.

☐ 1. Document ID: US 20030172109 A1

Using default format because multiple data bases are involved.

L3: Entry 1 of 9 File: PGPB Sep 11, 2003

PGPUB-DOCUMENT-NUMBER: 20030172109
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030172109 A1

TITLE: Trusted operating system

PUBLICATION-DATE: September 11, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Dalton, Christoper I.	Bristol		GB
Choo, Tse Huong	Bristol		GB
Norman, Andrew Patrick	Bristol		GB

US-CL-CURRENT: 709/203

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

☐ 2. Document ID: US 20030149895 A1

L3: Entry 2 of 9 File: PGPB Aug 7, 2003

PGPUB-DOCUMENT-NUMBER: 20030149895
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030149895 A1

TITLE: Trusted gateway system

PUBLICATION-DATE: August 7, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Choo, Tse Huong	Bristol		GB
Dalton, Christopher I	Bristol		GB
Norman, Andrew Patrick	Bristol		GB

US-CL-CURRENT: 713/167

☐ 3. Document ID: US 20030145235 A1

L3: Entry 3 of 9 File: PGPB Jul 31, 2003

PGPUB-DOCUMENT-NUMBER: 20030145235
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030145235 A1

TITLE: Network adapter management

PUBLICATION-DATE: July 31, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Choo, Tse Huong	Bristol		GB

US-CL-CURRENT: 713/167

☐ 4. Document ID: US 20030014466 A1

L3: Entry 4 of 9 File: PGPB Jan 16, 2003

PGPUB-DOCUMENT-NUMBER: 20030014466
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030014466 A1

TITLE: System and method for management of compartments in a trusted operating system

PUBLICATION-DATE: January 16, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Berger, Joubert	Atlanta	GA	US
Leerssen, Scott A.	Atlanta	GA	US
Choo, Tse Huong	Stoke Bristol		GB
Stock, Richard B.	North Somerset		GB
Dalton, Christopher I.	Redland Bristol		GB
Norman, Andrew Patrick	Stoke Bristol		GB

US-CL-CURRENT: 718/102

☐ 5. Document ID: US 20020143795 A1

L3: Entry 5 of 9 File: PGPB Oct 3, 2002

PGPUB-DOCUMENT-NUMBER: 20020143795
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020143795 A1

TITLE: Improved method and system of computer file management

PUBLICATION-DATE: October 3, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY
Fletcher, Thomas O.P.	Ottawa		CA
Van Der Veen, Peter H.	Kanata		CA
Dodge, Dan	Ottawa		CA

US-CL-CURRENT: 707/200

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

---

☐ 6. Document ID: US 6021408 A

L3: Entry 6 of 9 File: USPT Feb 1, 2000

US-PAT-NO: 6021408
DOCUMENT-IDENTIFIER: US 6021408 A

TITLE: Methods for operating a log device

DATE-ISSUED: February 1, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ledain; Joel E.	Santa Clara	CA		
Colgrove; John A.	Palo Alto	CA		
Koren; Dan	Incline Village	NV		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Veritas Software Corp.	Moutain View	CA			02

APPL-NO: 08/713204 [PALM]
DATE FILED: September 12, 1996

INT-CL: [06] G06 F 17/30

US-CL-ISSUED: 707/8; 707/201, 707/206, 707/202
US-CL-CURRENT: 707/8; 707/201, 707/202, 707/206

FIELD-OF-SEARCH: 395/888, 395/182.04, 341/50, 707/205, 707/202, 707/206, 707/201,

707/8, 711/114, 711/136, 711/161, 365/189.01

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5416915	May 1995	Mattson et al.	711/114
5418925	May 1995	DeMoss et al.	395/425
5448719	September 1995	Schultz et al.	395/182.03
5499367	March 1996	Bamford et al.	707/8
5530850	June 1996	Ford et al.	707/206
5537588	July 1996	Englemann et al.	707/202
5551003	August 1996	Mattson et al.	711/136
5553285	September 1996	Krakauer et al.	707/202
5557770	September 1996	Bhide et al.	711/161
5574952	November 1996	Brady et al.	395/888
5600596	February 1997	Shirakihara	365/189.01
5604902	February 1997	Burkes et al.	707/206
5644791	July 1997	Brady et al.	395/888
5659677	August 1997	Cohn et al.	395/182.04
5666114	September 1997	Brodie et al.	341/50
5666532	September 1997	Saks et al.	707/205
5870757	February 1999	Fuller	707/201

OTHER PUBLICATIONS

"Using Dual Actuator Shared Data Direct Access Storage Device Drives In a Redundant Array", IBM Technical Disclosure Bulletin, v33, n8, p270-272, Jan. 1991.
"Write-Only Disk Caches", Jon A. Solworth & Cyril U. Orji, ACM, 1990, pp. 123-132.
"Disk Subsystem Load Balancing: Disk Striping vs. Conventional Data Placement", Ganger, et al., IEEE, 1993, pp. 40-49.
"An Implementation of a Log-Structured File System for UNIX", Staelin, et al., Hewlett-Packard, Nov. 30, 1992.
"The Logical Disk: A New Approach to Improving File Systems", Jonge, et al., ACM, 1993, pp. 15-28.
"Strategies to Improve I/O Cache Performance", Richardson, et al., IEEE, 1993, pp. 31-39.
"Understanding Background Data Movement in a Log-Structured Disk Subsystem", Bruce McNutt, IBM Journal of Research and Development, 1994.
"The LFS Storage Manager", Rosenblum, et al., '90 USENIX Technical Conference, Jun., 1990.
"The Design and Implementation of a Log-Structured File System", Rosenblum, et al., Proceedings of the 13th ACM Symposium on Operating Systems Principles, Jul. 24, 1991.
"Beating the I/O Bottleneck: A Case for Log-Structured File Systems", Ousterhout, et al., Computer Science Division, Electrical Engineering & Computer Sciences, University of California at Berkeley, Oct. 30, 1988.

ART-UNIT: 271

PRIMARY-EXAMINER: Black; Thomas G.

ASSISTANT-EXAMINER: Rones; Charles L.

ATTY-AGENT-FIRM: Rosenberg; Gerald B. New Tech Law

ABSTRACT:

A log device is coupled in the logical data transfer path between a storage device, which provides for the storage of file and system data within a main filesystem layout, and a computer system. The log device provides for the storage of the file and system data within a log structured filesystem layout. A control program is executed to manage the storage of file and system data in data segments in the log device filesystem and to selectively transfer the file and system data from the log device to the storage device. The control program utilizes location data provided in the file and system data to identify a destination storage location for the file and system data within the main filesystem layout.

18 Claims, 16 Drawing figures

☐ 7. Document ID: US 5996054 A

L3: Entry 7 of 9 File: USPT Nov 30, 1999

US-PAT-NO: 5996054
DOCUMENT-IDENTIFIER: US 5996054 A

TITLE: Efficient virtualized mapping space for log device data storage system

DATE-ISSUED: November 30, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ledain; Joel E.	Santa Clara	CA		
Colgrove; John A.	Palo Alto	CA		
Koren; Dan	Incline Village	NV		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Veritas Software Corp.	Mountain View	CA			02

APPL-NO: 08/711890 [PALM]
DATE FILED: September 12, 1996

INT-CL: [06] G06 F 13/00

US-CL-ISSUED: 711/203; 711/6, 711/112, 711/207, 711/206, 709/301
US-CL-CURRENT: 711/203; 711/112, 711/206, 711/207, 711/6, 719/321

FIELD-OF-SEARCH: 711/112, 711/113, 711/114, 711/4, 711/144, 711/207, 711/203, 711/6, 395/182.04, 395/182.03, 395/182.09, 395/872, 395/182.13

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5297258	March 1994	Hale et al.	395/275
5418925	May 1995	DeMoss et al.	395/425
5448719	September 1995	Schultz et al.	395/182.03
5551003	August 1996	Mattson et al.	711/136
5553285	September 1996	Krakauer et al.	395/600
5636360	June 1997	Courts et al.	395/472
5666532	September 1997	Saks et al.	707/205
5732238	March 1998	Sarkozy	395/440
5754888	May 1998	Yang et al.	395/872
5832515	November 1998	Ledain et al.	707/202

OTHER PUBLICATIONS

"Write-Only Disk Caches", Jon A. Solworth & Cyril U. Orji, ACM, 1990, pp. 123-132.
"Disk Subsystem Load Balancing: Disk Striping vs. Conventional Data Placement", Ganger, et al., IEEE, 1993, pp. 40-49.
"An Implementation of a Log-Structured File System for UNIX", Staelin, et al., Hewlett Packard, Nov. 30, 1992.
"The Logical Disk: A New Approach to Improving File Systems", Jonge, et al., ACM, 1993, pp. 15-28.
"Strategies to Improve I/O Cache Performance", Richardson, et al., IEEE, 1993, pp. 31-39.
"Understanding Background Data Movement in a Log-Structured Disk Subsystem", Bruce McNutt, IBM Journal of Research and Development, 1994.
"The LFS Storage Manager", Rosenblum, et al., '90 USENIX Technical Conference, Jun., 1990.
"The Design and Implementation of a Log-Structured File System", Rosenblum, et al., Proceedings of the 13th ACM Symposium on Operating Systems Principles, Jul. 24, 1991.
"Beating the I/O Bottleneck: A Case for Log-Structured File Systems", Ousterhout, et al., Computer Science Division, Electrical Engineering & Computer Sciences, University Of California at Berkeley, Oct. 30, 1988.
Tzi-cker Chiueh, Trail: A Track-based Logging Disk Architecture for Zero-Overhwead Writes, pp. 339-343, 1993.

ART-UNIT: 272

PRIMARY-EXAMINER: Cabeca; John W.

ASSISTANT-EXAMINER: Bataille; Pierre-Michel

ATTY-AGENT-FIRM: Rosenberg; Gerald B. New Tech Law

ABSTRACT:

A log device based data storage subsystem provides for the efficient storage and retrieval of data with respect to an operating system executing on a computer system coupled to the data storage system. The data storage system includes a

storage device providing for the storage of predetermined file and system data, as provided by the computer system, within a main filesystem layout established in the storage device. The data storage system also includes a log device coupled in the logical data transfer path between storage device and the computer system. The log device provides for the storage of the predetermined file and system data within a log structured filesystem layout established in the log device. A control program, included as part of the data storage system, is executed in connection with the log device and provides log structured filesystem management over the log device to store the predetermined file and system data in one of a plurality of data segments, delimited by a first free data segment and an oldest filled data segment, to selectively clean the oldest filled data segment to the first free data segment, and to selectively transfer the predetermined file and system data from the log device to the storage device. The control program utilizes location data provided in the predetermined file and system data to identify a destination storage location for the predetermined file and system data within the main filesystem layout.

10 Claims, 16 Drawing figures

☐ 8. Document ID: US 5832515 A

L3: Entry 8 of 9 File: USPT Nov 3, 1998

US-PAT-NO: 5832515
DOCUMENT-IDENTIFIER: US 5832515 A

TITLE: Log device layered transparently within a filesystem paradigm

DATE-ISSUED: November 3, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ledain; Joel E.	Santa Clara	CA		
Colgrove; John A.	Palo Alto	CA		
Koren; Dan	Incline Village	NV		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Veritas Software	Mt. View	CA			02

APPL-NO: 08/711889 [PALM]
DATE FILED: September 12, 1996

INT-CL: [06] G06 F 17/30

US-CL-ISSUED: 707/202; 395/681
US-CL-CURRENT: 707/202; 719/321

FIELD-OF-SEARCH: 395/681, 395/618, 395/619, 395/621, 395/622, 395/438-441, 395/497.01, 395/497.04, 707/202, 707/203, 707/205, 707/206

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
5418925	May 1995	DeMoss et al.	395/425
5448719	September 1995	Schultz et al.	395/182.03
5488701	January 1996	Brady et al.	395/182.04
5530850	June 1996	Ford et al.	395/622
5537588	July 1996	Engelmann et al.	395/618
5550998	August 1996	Willis et al.	395/441
5551003	August 1996	Mattson et al.	395/463
5555371	September 1996	Duyanovich et al.	395/182.11
5600596	February 1997	Shirakihava	365/189.01
5666114	September 1997	Brodie et al.	341/50
5666532	September 1997	Saks et al.	395/621

OTHER PUBLICATIONS

Tom Yager, "The Great Little File System," Byte, Feb. 1995, 4 pages at http://www.byte.com/art/9502/sec13/art2.htm.
Mike Holton et al., "XFS: A Next Generation Journaled 64-Bit Filesystem With Guaranteed Rate I/O," XFS White Paper, Silicon Graphics, Inc., pp. 1-19, date unknown.
"Write-Only Disk Caches", Jon A. Solworth & Cyril U. Orji, ACM, 1990, pp. 123-132.
"Disk Subsystem Load Balancing: Disk Striping vs. Conventional Data Placement", Ganger, et al., IEEE, 1993, pp. 40-49.
"An Implementation of a Log-Structured File System for UNIX", Staelin, et al., Hewlett Packard, Nov. 30, 1992, 20 pages.
"The Logical Disk: A New Appraoch to Improving File Systems", Jonge, et al., ACM, 1993, pp. 15-28.
"Strategies to Improve I/O Cache Performance", Richardson, et al., IEEE, 1993, pp. 31-39.
"Understanding Background Data Movement in a Log-Structured Disk Subsystem", Bruce McNutt, IBM Journal of Research and Development, 1994, 8 pages.
"The LFS Storage Manager", Rosenblum, et al., '90 USENIX Technical Conference, Jun., 1990, pp. 1-15.
"The Design and Implementation of a Log-Structured Filed System", Rosenblum, et al., Proceedings of the 13th ACM Symposium on Operating Systems Principles, Jul. 24, 1991, pp. 1-15.
"Beating the I/O Bottleneck: A Case for Log-Structured File Systems", Ousterhout, et al., Computer Science Division, Electrical Engineering & Computer Sciences, University Of California at Berkeley, Oct. 30, 1988, pp. 1-17.

ART-UNIT: 271

PRIMARY-EXAMINER: Kulik; Paul V.

ATTY-AGENT-FIRM: Fliesler, Dubb, Meyer & Lovejoy

ABSTRACT:

A log device based data storage subsystem provides for the efficient storage and

retrieval of data with respect to an operating system executing on a computer system coupled to the data storage system. The data storage system includes a storage device providing for the storage of predetermined file and system data, as provided by the computer system, within a main filesystem layout established in the storage device. The data storage system also includes a log device coupled in the logical data transfer path between storage device and the computer system. The log device provides for the storage of the predetermined file and system data within a log structured filesystem layout established in the log device. A control program, included as part of the data storage system, is executed in connection with the log device and provides log structured filesystem management over the log device to store the predetermined file and system data in one of a plurality of data segments, delimited by a first free data segment and an oldest filled data segment, to selectively clean the oldest filled data segment to the first free data segment, and to selectively transfer the predetermined file and system data from the log device to the storage device. The control program utilizes location data provided in the predetermined file and system data to identify a destination storage location for the predetermined file and system data within the main filesystem layout.

27 Claims, 16 Drawing figures

☐ 9. Document ID: US 6021408 A

L3: Entry 9 of 9 File: DWPI Feb 1, 2000

DERWENT-ACC-NO: 2000-146984
DERWENT-WEEK: 200013

TITLE: Method for data storage in a log device

INVENTOR: COLGROVE, J A; KOREN, D ; LEDAIN, J E

PRIORITY-DATA: 1996US-0713204 (September 12, 1996)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 6021408 A	February 1, 2000		023	G06F017/30

INT-CL (IPC): G06 F 17/30

ABSTRACTED-PUB-NO: US 6021408A
BASIC-ABSTRACT:

NOVELTY - A separate device driver (44) is selectively coupled between the computer system filesystem module (32) and the first device driver (34) of a storage device for transferring data blocks from a second storage device to the storage device. The data blocks are read from the first storage device.

DETAILED DESCRIPTION - Data blocks from the file system module are synchronously written into a data segment on a second storage device, a map of the data blocks is then created that can be identified by the second device driver. The data blocks

are transferred from the second storage device to the first storage device by using the second device driver independent of the filesystem module.

USE - Filesystem design for log devices such as disk drives in a computer systems.

ADVANTAGE - The operation of the and management of the log device is independent from the main filesystem and mass storage subsystem allowing the devices to be optimized independently.

DESCRIPTION OF DRAWING(S) - The drawing shows a data flow diagram for implementing a log device.

Filesystem module 32

Device driver 34

Second device driver 44

Term	Documents
COMPUTER	1567437
COMPUTERS	336120
SYSTEM	6666894
SYSTEMS	2304527
(2 AND (COMPUTER NEAR SYSTEM)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	9
(L2 AND (COMPUTER NEAR SYSTEM)).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	9

L3: Entry 6 of 9 File: USPT Feb 1, 2000

DOCUMENT-IDENTIFIER: US 6021408 A
TITLE: Methods for operating a log device

Abstract Text (1):
A log device is coupled in the logical data transfer path between a storage device, which provides for the storage of file and system data within a main filesystem layout, and a computer system. The log device provides for the storage of the file and system data within a log structured filesystem layout. A control program is executed to manage the storage of file and system data in data segments in the log device filesystem and to selectively transfer the file and system data from the log device to the storage device. The control program utilizes location data provided in the file and system data to identify a destination storage location for the file and system data within the main filesystem layout.

Brief Summary Text (3):
The present invention is generally related to high-performance computer filesystem designs used in conjunction with contemporary operating systems and, in particular, to a multi-tasking computer system employing a log device to support a log structured filesystem paradigm over an independent filesystem and the operation of the log device to dynamically balance filesystem I/O transactions.

Brief Summary Text (5):
In the operation of conventional computer systems, the overall performance of the system is often constrained by the practically achievable throughput rates of secondary mass storage units, typically implemented variously utilizing single disk drives and cooperatively organized arrays of disk drives. As the peak performance of central processing units has dramatically increased, performance constraints have significantly increased due to the relatively lesser advances in performance achievable by secondary mass storage units. Factors that affect the performance of disk drive type devices include, in particular, the inherent mechanical operation and geometric relations imposed by the fundamental mechanical construction and operation of conventional disk drives. The essentially sequential operating nature of disk drives and the extremely disparate rates of data read/write, actuator seek rates and rotational latencies result in the performance of secondary mass storage devices being highly dependant on the layout and logical organization of data on the physical data storage surfaces of disk drives.

Brief Summary Text (7):
Another factor that can significantly influence the optimum layout of data is the nature of the software applications executed by the central processing unit at any given time. Different optimizations can be effectively employed depending on whether there is a preponderance of data reads as compared to data writes, whether large or small data block transfers are being performed, and whether physical disk accesses are highly random or substantially sequential. However, the mix of concurrently executing applications in most computer systems is difficult if not practically impossible to manage purely to enforce disk drive operation optimizations. Conventionally, the various trade-offs between different optimizations are statically established when defining the basic parameters of a filesystem layout. Although some filesystem parameters may be changeable without re-installing the filesystem, fundamental filesystem control parameters are not

changeable, and certainly not dynamically tunable during active filesystem operation.

Brief Summary Text (8):
An early effort to improve the performance of secondary mass storage devices involved providing a buffer cache within the primary memory of the computer system. Conventional buffer caches are logically established in the file read/write data stream. Repeated file accesses and random accesses of a particular file close in time establish initial image copies of the file contents within the buffer cache. The subsequent references, either for reading or writing, are executed directly against the buffer cache with any file write accesses to the mass storage subsystem delayed subject to a periodic flushing of write data from the buffer cache to secondary mass storage. The buffer cache thus enables many file read and write operations to complete at the speed of main memory accesses while tending to average down the peak access frequency of the physical secondary mass storage devices.

Brief Summary Text (9):
A significant drawback of merely using a buffer cache to improve overall system performance arises in circumstances where data integrity requirements require write data to be written to a non-volatile store before the write access can be deemed complete. In many networked computer system applications, particularly where connectionless communication protocols are utilized for file data transport over the network, the requirement that file write data accesses be completed to non-volatile store is a fundamental requirement of the network protocol itself. Thus, conventionally, the file access latencies incurred in writing data to secondary mass storage devices are a component of and compounded by the latencies associated with data transport over both local and wide area networks.

Brief Summary Text (14):
In addition to the practical issues associated with using a disk drive as a cache memory, logical data management problems are also encountered. Preferably, the file read data stream is logically routed around the disk cache and supported exclusively through the operation of the RAM buffer cache. File write data bypasses the main memory buffer cache and is written exclusively to the disk cache. Particularly in multi-user a networked computer system environments, multiple independent read and write file accesses may be directed against a single file within a rather small time frame. Since the requests are ultimately associated with potentially independent processes and applications, the computer operating system or at least the subsystem managing the buffer and disk caches must provide a mechanism for preserving data integrity. Write data requests must be continually resolved against prior writes of the same block of file data as stored by the disk cache. Each read of a file data block must be evaluated against all of the data blocks held by the write disk cache. While many different bypass mechanisms and data integrity management algorithms have been developed, the fundamental limitation of a disk cache remains. Repeated accesses to the disk cache are required not only in the ordinary transfer of write file data to the cache but also in management of the cache structure and in continually maintaining the current integrity of the write file data stream. Consequently, the potential performance improvements achievable by a disk cache are further limited in practice.

Brief Summary Text (20):
The general acceptance of log structured filesystems for certain, typically write intensive, computer applications reflects the significant improvement available through the use of a direct sequential write log structure filesystem. The available write data bandwidth, even in the presence of continuing log cleaning and maintenance operations, can be near or above 70 percent. In addition, log structured filesystems provides a number of ancillary benefits involving the reduced latency of atomic file data write operations and improved data integrity verification following computer system crashes. Particularly in network support

related operations, the direct writing of write file data to a log structured filesystem, including directory related information as an essentially atomic operation minimizes the total latency seen in completing committed write network data write transfer operations. Similarly, by virtue of all write file data operations being focused at the end of the active log, as opposed to being scattered throughout the disk drive storage space, data verification operations need only be focused on evaluating just the end of the log end rather than the entire directory and data structures of a conventional filesystem. Consequently, both network data write operations and the integrity of all data written is improved by writing directly to a permanent log structured filesystem.

Brief Summary Text (21):
Log structured filesystems are, however, not entirely effective in all computing environments. For example, log structured filesystems show little improvement over conventional filesystems where the computing environment is subject to a large percentage of fragmentary data writes and sequential data reads such as may occur frequently in transactional data base applications. The write data optimizations provided by log structured filesystems can also be rather inefficient in a variety of other circumstances as well, such as where random and small data block read accesses are dominant. Indeed, as computer systems continue to grow in power and are required to support more and different application environments concurrently with respect to a common mass storage subsystem, the tension between applications for optimal use of the disk drive bandwidth provided by the mass storage system will only continue to increase.

Brief Summary Text (25):
This is achieved through the use of a data storage subsystem that provides for the efficient storage and retrieval of data with respect to an operating system executing on a computer system coupled to the data storage system. The data storage system includes a storage device providing for the storage of predetermined file and system data, as provided by the computer system, within a main filesystem layout established in the storage device. The data storage system also includes a log device coupled in the logical data transfer path between storage device and the computer system. The log device provides for the storage of the predetermined file and system data within a log structured layout established in the log device. A control program, included as part of the data storage system, is executed in connection with the log device and provides system management over the log device to store the predetermined file and system data in one of a plurality of data segments, delimited by a first free data segment and an oldest filled data segment, to selectively clean the oldest filled data segment to the first free data segment, and to selectively transfer the predetermined file and system data from the log device to the storage device. The control program utilizes location data provided in the predetermined file and system data to identify a destination storage location for the predetermined file and system data within the main filesystem.

Brief Summary Text (26):
An advantage of the present invention is that the operation and management of the log device is independent of the operation and management and indeed the layout of the main filesystem of the mass storage subsystem. The independent operation and management of the log device allows the operation and management of the main filesystem to be optimized independently of the log device to best serve optimization typically for read accesses, such as by file and directory clustering and various forms of data striping to improve the integrity and logical survivability of the mass storage device.

Brief Summary Text (30):
Still another advantage of the present invention is that the operation and management of the log device allows for transparent compression and other data transforms, such as encryption and data hardening, to be selectively applied to the log data as stored in the log structured filesystem of the log device. In

particular, high-speed compression of data to be stored on the log device permits a larger effective main filesystem space to be logged, thereby improving the effective read performance available from the main filesystem while maintaining the improved write performance provided by the log device.

Brief Summary Text (31):
Yet still another advantage of the present invention is that main filesystem directory information is encoded into system data that is stored with file data within the data segments stored by the log device. This encoded information permits high performance data location mapping algorithms to provide for translation and inverse translation of log device stored block numbers while minimizing the use of the main memory of a computer system for storing the translation maps. A strategy of progressive writing of partial translation maps and related data to the log device in conjunction with or as part of data segments ensures that the log device is robust and easily recoverable in the event of system crashes and the like.

Brief Summary Text (32):
A still further advantage of the present invention is that the log device itself may be physically structured as a mirrored or RAID based disk drive subsystem operating from the same or a different disk drive controller as the main filesystem storage devices. The reliability of the log device and recoverability of the data stored by the log device can be readily scaled to match appropriately the speed and integrity capabilities of the main filesystem storage device.

Brief Summary Text (33):
A yet still further advantage of the present invention is that the independent operation of the log device permits independent management operations to be performed on the main filesystem storage device in a dynamic manner. Through the operation of the log device, the fundamental filesystem structure of the main filesystem storage device can be changed between a mirrored and arrayed structure and between different levels of RAID structure through progressive management of alterations in the translations maps relating logical and physical storage of file and system data by the main filesystem storage device.

Brief Summary Text (34):
Still another advantage of the present invention is that the log device may be implemented and operated without main filesystem disks actually being used. Where data writes may greatly predominate and particularly where the written data has a relatively short valued life span, write data can be efficiently directed to the log device for essential transient storage and then expired when no longer needed.

Drawing Description Text (3):
FIG. 1 is a basic block diagram of a computer system implementing a log device in accordance with the present invention;

Drawing Description Text (4):
FIG. 2 is a data flow diagram illustrating the effective insertion of the log device pseudo-device driver into the lower levels of the operating system to intercept and selectively route file and system data with respect to the log device disk and through the main disk device driver with respect to the main filesystem storage device;

Detailed Description Text (2):
A computer system 10 including a preferred embodiment of the present invention is shown in FIG. 1. A host processor or central processing unit (CPU) 12 connects through a system bus 14 to a main memory 16 that is used to store application programs and data as well as an operating system used to supervise execution of the applications. The operating system conventionally includes a filesystem module, such as the conventional UNIX.RTM. File System (UFS), and device drivers suitable for establishing the operative interface between the operating system and various

peripherals accessible by the CPU 12 through the system bus 14. Optionally, a non-volatile RAM block (NVRAM) 18 can be provided as an adjunct to the main memory 16 for storing certain relatively critical data that may be advantageously utilized during the operation of the computer system 10 and potentially during recovery from a system crash condition.

Detailed Description Text (6):
A data flow diagram 30, showing modified and generally preferred data transport paths in a preferred embodiment of the present invention, is presented in FIG. 2. An operating system core 32 including a file system module, collectively referred to as a Unix kernel, exchanges device independent data streams with a main disk device driver 34 via data paths 36, 38. The main disk device driver 34 provides the necessary and conventional support for transferring the device independent data streams to the main filesystem disks 40 in a manner consistent with the device dependencies necessary to secure data storage and retrieval with respect to the main filesystem disks 40.

Detailed Description Text (7):
In accordance with the preferred embodiment of the present invention, a log device pseudo-device driver 44 is provided in connection with the operating system core 32 and main disk device driver 34. Preferably, the data stream exit and entry points of the operating system 32, otherwise connected to the paths 36, 38, are instead routed via paths 42, 50 to the log device pseudo-device driver 44. The data stream entry and exit points of the main disk device driver are also coupled to the log device pseudo-device driver 44 via new data transfer paths 46, 48. Thus, at least data that is to be written to or read from a selected filesystem nominally maintained on the main filesystem disks 40 is routed through the log device pseudo-device driver 44 and may be made subject to the control operations established by the execution of the log device pseudo-device driver 44. Specifically, the log device pseudo-device driver 44 selectively provides for the routing of filesystem data directed to the main filesystem disks 40 to be at least temporarily stored and potentially read back from the log device disks 52.

Detailed Description Text (8):
By the particular construction of a separate specialized file data layout on the log device disks 52 that is independently operated under control of the log device pseudo-device driver 44, the present invention provides for a highly throughput optimized apparent filesystem write data transfer path from the operating system core 32 through the log device pseudo-device driver 44 to the log device disk 52. This optimized write data path through the log device pseudo-device driver 44 is effectively independent of any read data path 38 that may selectively remain for other logical devices supported by the main disk device driver 34. The read data path 48, 50 from the main file system disks 40 through the log device pseudo-device driver 44 is only slightly less independent of any active write data path to the log device pseudo-device driver 44 due to any potential competition in execution of the log device pseudo-device driver 44 itself. The write data path 42 through the log device pseudo-device driver 44 to the log device disk 52 will be co-dependant on the concurrent use of the read data path from the log device disk 52 through the log device pseudo-device driver 44 to the operating system core 32 via the path 50. The significance of this co-dependance is mitigated by the preferred utilization of a conventional buffer cache for servicing repeated reads within the operating system core 32 itself and substantially avoiding the re-reading of data very recently written to the log device disks 52. Thus, by the preferred operation of the present invention, relatively infrequently read data stored by a logged filesystem, i.e., a filesystem provided on the main filesystem disks that is subject to the logging operation of the log device pseudo-device driver 44, is predominately serviced from the main filesystem disks 40.

Detailed Description Text (9):
In general, all write data directed to a logged filesystem on the main filesystem

disks 40 is written to the log device disks 52. A number of selective exceptions and conditions may be recognized by the log device pseudo-device driver 44 where logged filesystem write data may be nonetheless written directly to the main filesystem disks 40. The desired type or types of data that are stored by the log device is preferably a dynamically alterable characteristic tuned in light of the particular nature and use of the logged filesystem. Where read and write data is generally small and random, all write data, including meta-data, may be written to the log device disk 52. If user data writes are typically large or determined to exceed a programmable size threshold, the log device pseudo-device driver 44 may direct such writes directly to the logged filesystem. Small meta-data write and perhaps small user data writes might then be staged to the log device disks pending a large write migration of such data to the logged filesystem itself. In some situations, the predominant component of the write data for a logged filesystem may be a transaction log or the like. As an essentially write only data object, the write data of a transaction log might alone be directed to the log device disks 52 and later migrated to archival storage or simply discarded as appropriate for the use of the transaction log. Indeed, anytime where data reads greatly predominate data writes, and the volume of new data writes, as opposed to data overwrites, relative to the size of the log device disks 52 is small, the log device disks 52 can be used stand-alone. That is, while the operating system core 32 fully implements a filesystem for managing the storage of data on the main filesystem disks 40, the complete data storage function is performed by the log device disks 52 alone. Physical main filesystem disks 40 may not actually be implemented. The existence and operation of the log device pseudo-device driver 44 is transparent to the filesystem paradigm implemented by the operating system core 32. Consequently, the present invention allows a wide spectrum of log device usage strategies to be constructed and implemented in or through the log device pseudo-device driver 44 based on simple, easily distinguished characteristics of particular data being written to a logged filesystem.

Detailed Description Text (10):
A cleaner daemon is periodically executed under the control of the operating system core 32 to progressively clean main filesystem data as stored on the log device disk 52 and, subject to a number of constraint conditions, provide for the migration of such filesystem data from the log device disk 52 through the main disk device driver 34 for final storage on the main filesystem disks 40. Thus, reads of recently written data that can no longer be satisfied through the use of the buffer cache and that has not been cleaned to the main filesystem disks 40 are satisfied from the log device disks 52.

Detailed Description Text (11):
Organization of the data stored by the log device disk 52 in a physical layout independent of the filesystem format of the main filesystem disks provides a number of advantages over other data organizations including disk caches and stream buffers. The establishment of an ordered data layout on the log device disks 52 enables stored data to be readily recovered in the event of a system crash. All completed atomic transfers of data to the log device disks 52 are fully recoverable from the log device disks 52 without necessary reference to any control data as may, for example, be stored transiently in main memory 16 or at significant cost, in the NVRAM memory 18, at the time of the crash. Each atomic write transaction of user data to the log device disks 52 includes, by operation of the log device pseudo-device driver 44, encoded system data, also referred to as meta-data and distinguished from user data, that can be subsequently decoded to determine the intended destination of the data within a specific logged filesystem on the main filesystem disk 40. As a consequence, the resident store of completed atomic transactions held by the filesystem on the log device disks 52 need only be progressively examined during a system recovery operation to determine where valid data found on the log device disks 52 is to be stored in a corresponding logged filesystem on the main filesystem disks 40. The progressive examination need only proceed from the last known good data checkpoint in the operation of the log device

disks 52. The encoded location data enables the performance of data migration from the log device disk 52 to the main filesystem disk 40 both in reconstruction of the main filesystem of the computer system 10 and simplifies execution of the background cleaning process.

Detailed Description Text (12):
In addition, encoding and storing system location data with the user data as stored in data segments in the filesystem of the log device disks 52 permits the data segments to be manipulated and relocated within the filesystem on the log device disks 52. Both data relocation and block compaction can be performed by the cleaner process on an ongoing basis independent of the writing of new data segments to the filesystem on the log device disks 52 and the subsequent migration of data to the logged filesystem on the main filesystem disks 40. Thus, the segment storage space represented by the log device disk 52 is continually cleaned and optimized to receive immediate writes of data segments from the operating system 32 and to selectively defer migration of data to the main filesystem disks 40 in optimization of filesystem data reads by the operating system core 32.

Detailed Description Text (14):
As reflected in FIG. 2, a main disk device driver 54 is conventionally provided to support the main filesystem device dependencies. These device dependencies are relatively conventional and typically provide for the establishment of I/O control, raw, and buffered data stream interfaces to the device driver interface of the operating system core 62. Typically, and preferably, the operating system core 62 includes a conventional device independent Unix filesystem (UFS) or equivalent component that establishes the logical layout of a filesystem maintained on the main filesystem disks 40. The interface presented by the operating system 62 to any relevant attached device drivers includes open, close, read, write and I/O control (IOCTL) interface points, as appropriate for both raw character data streams and, in particular, buffered block data streams. These interface points route through character and block device switch tables that serve to direct the potentially many different data streams supported by the operating system to one or more resident device drivers. Specifically, the data streams are associated with logical major and minor device numbers that are correlated through the switch tables to specify a corresponding device driver entry point to call for the transfer a particular stream of character or block data with respect to the operating system core 62. The switch tables thus effectively operate to selectively map the device driver interface of the operating system core 62 to a specific corresponding interface of a device driver. The device switch tables are initially constructed in connection with the initialization of each of the device drivers associated with the operating system core 62. The device driver initialization routines are effectively discovered and executed by the operating system core 62 during global core operating system initialization.

Detailed Description Text (15):
The device driver initialization routine provided effectively within the IOCTL interface 64 of the log device pseudo-device driver 61 is called subsequently to the execution of the main disk device driver 34 initialization routine. With the device switch tables already initialized with entry points for the main disk device driver, the initialization routine of the pseudo-device driver selectively calls an I/O steal routine 68 that operates to remap selected device driver pointer entries within the device switch tables. A configuration data file stored within the main filesystem is read at the request of the I/O steal routine 68. The configuration file, preferably designated as /etc/dx.conf, supplies an identification and any applicable qualifications on the intercession of the log disk pseudo-device driver with respect to a specific potentially logged filesystem provided on the main filesystem disks 40.

Detailed Description Text (21):
In a preferred embodiment of the present invention, the log device pseudo-device

driver distinguishes between native IOCTL commands, issued by the operating system core 62 for execution by the main filesystem device driver 34, and local IOCTL commands that are to be executed by the log device pseudo device driver 61. The native disk driver IOCTL commands intercepted through the remapped, or stolen, device switch table entry points are handled in three distinct ways, depending on their intended function. Native IOCTL commands that retrieve or set device specific information, such as device state or geometry, are passed through to the main filesystem device driver 34 for conventional execution. IOCTL commands that conflict with the logging of a main filesystem are rejected, typically with ENXIO return codes. Lastly, IOCTL commands that perform special read or write functions, such as to detect and resolve the existence of inconsistent log device disk mirrors, are processed through a log device data block location translation algorithm and executed by the main filesystem device driver 61 against the disks of the log device.

Detailed Description Text (30):
A Sync Logged Device (LOG.sub.-- DEV.sub.-- SYNC) IOCTL command instructs the log device pseudo-device driver 61 to set the state of a logged device to synchronized (sync'd). This command does not actually perform any data movement to sync logged data back to the main filesystem disks. Rather a separate user mode utility program is used to suspend logging by a log disk relative to a specified main filesystem and to flush corresponding currently logged data to that main filesystem. The Sync Logged Device IOCTL command is then issued by the user mode utility to obtain confirmation of the completion of the sync operation or to receive an EBUSY return code if any logged data remains in the log device for the specified logged filesystem.

Detailed Description Text (53):
The log manager routine 74 interoperates with the log map routine 72 and the segment I/O routines 78 to implement the physical log structure layout established on the log device. The log map routine 72 manage a number of translation maps ultimately utilized to establish logical correlations between data blocks, as stored in log blocks within data segments, all as stored on the log device, and the data block storage locations within the filesystem layout structure established for the logged filesystem on the main filesystem disks 40. These log maps permit data blocks to be referenced by the log device pseudo-device driver 61 based on logical references specific to the organization of the file system established on the main filesystem disks. By maintaining the appearance of a single location representation for all user data that is passed through the log device pseudo-device driver 61, independent and even multiple filesystem organizations on the main filesystem disks 40 can be supported through the log device transparently with respect to the kernel mode operating system core 62 itself.

Detailed Description Text (61):
The monitor routines 86 collect a variety of statistics concerning principally the effective performance of data transfers performed by the I/O steal routine 68. The monitored statistics preferably allow for a direct analysis of the number, size and type of data block transfers and data segment transfers performed through calls to the I/O steal routines 68. The collected information permits performance analysis on both a per request basis and a unit time basis. Particular statistics monitored include the rate of and relative proportion of read and write requests received through the IOCTL interface 64. Also monitored are the relative rates of reads and writes of data segments to the log device disks 52, the frequency that read requests are satisfied from the log device as opposed to the main filesystem disks 40, and the effective rate of compaction of data segments as log tail data segments are cleaned and new data segments are written to the log head.

Detailed Description Text (63):
The trace records and operational statistics collected by the volume trace routines 84, monitor routines 86, and event trace routines 88 are preferably accessible by

both user and kernel mode daemons 90 executing under the control of the kernel mode operating system core 62. The event traces and information received by the daemons 90 is preferably used as the basis for dynamic analysis and adjustment of certain aspects of the on-going operation of the log device pseudo-device driver 61. Dynamic reconfiguration or tuning of fundamental operational parameters affecting the operation of the log device pseudo-device driver 61 and the structure of data segments as written out to the log device disks 52 is preferably directed by one or more daemons 90 through operating system calls to the kernel mode operating system core 62, resulting in the issuance of IOCTL commands to the IOCTL interface 64. Thus, daemons 90 may operate to implement any of a number of different log device specific operational strategies and further dynamically vary these strategies specifically in view of the analyzed performance of the log device pseudo-device driver 61. Although the fundamental parameters of a main filesystem layout may be conventionally static, dynamic tuning of the log disk performance allows an optimization of both the write data efficiency to the log device and the read data efficiency from the main filesystem disks. Consequently, as application load characteristics concerning the size, type and frequency of data reads and writes changes, the configuration of the log device pseudo-device driver 61 can be continually adjusted to obtain and maintain optimum performance.

Detailed Description Text (66):
The device number is initially translated through the use of a device table 102 that serves to map kernel mode operating system core provided device numbers to a consecutive set of small integers that the log device pseudo-device driver 61 utilizes internally to separately identify logged filesystem established within the main filesystem disks. At present, the internal device numbers are limited to four bits each. Thus, up to 15 separate filesystems may be logged through a singular instance of the log device pseudo-device driver 61. A copy of the contents of the device table 102, including device name, device number and internal device number are preferably saved as part of the superblock on the log device disks 108. A typically initialization time execution of the log device superblock manager 70 performs the initial superblock read and validation. The initialization routine then restores from the superblock the contents of the device table 102 for subsequent use by the log map routines 72. In addition, an IOCTL command to initiate the logging of a new filesystem on the main filesystem disks 40, or to detach a previously logged filesystem results in a corresponding update of the device table 102 by the log map routines 72. The new contents of the device table 102 written out to the log device with the next update of the log device superblock by the log device superblock manager 70.

Detailed Description Text (67):
The internal device number and the block number are then provided as inputs to a range compression algorithm 104 that places the significant information provided by the internal device number and block number into a single word wide 32-bit value. In accordance with the present invention, the block number can be scaled down, or right shifted, where the basic data block size used by the log device is larger than that of the logged file system on the main file system disks. Typical block sizes for the main filesystem disks 40 are 512 bytes and 1 kilobyte (kbyte). Log block sizes may range from 512 bytes to 8 kbytes or more.

Detailed Description Text (73):
When a request is received to read a data block from a main filesystem whose entry points have been stolen, the log device pseudo-device driver 61 is required to determine whether the requested data block is currently stored on the log device disks 52. Where, the corresponding logged filesystem has at least not been detached, the translation algorithm 100 is utilized to identify the potentially corresponding log device drive and log block from the segment table 112. If the successively identified entries in the super map, and map cache are empty or invalid, the lookup fails and the read request must be satisfied from the main filesystem disks 40. If the lookup succeeds through to the segment table, the

selected segment table entry is examined. Each used and valid segment table entry stores a device number and block number entry for each data block stored within the data segment. The data blocks within the data segment and the device/block numbers in the segment table entry are stored with an ordered correspondence. Other similarly ordered status bits stored by the segment table entry specify whether a corresponding data block is used and valid. Thus, a data block specifically referenced by the operating system core 62 can be determined to be stored on the log device and, further, distinguished as being presently valid or invalid before the data segment need be read in by the log device pseudo-device driver. Since data segments are of known size, the log block number can be used, subject to a modulo data segment size calculation, to identify the data segment containing the addressed log block.

Detailed Description Text (74):
Where the identified data block is marked valid, then at least the relevant log block of the data segment can be read in by operation of the segment I/O routine 78 and the referenced data block transferred through the data interface 66 to the operating system core 62. Alternately, where the data block is to be migrated or flushed out to the main filesystem disks 40, the device and block number for the data block is passed to and used by the main disk device driver 34 to specify the transfer destination of the data block on the main filesystem disks 40.

Detailed Description Text (79):
the ASCII device names and device major/minor numbers of the main filesystem devices that are being logged, including information for each defining the relevant state of the filesystem (logged, suspended) and configuration data for the volume, monitor and event routines;

Detailed Description Text (86):
Within the segment data block, as shown in FIG. 5c, the data segments are arrayed as a continuous sequence of data segments distinguished with respect to the use of the log device disk as including a physically first segment, a physically last segment, and a current first free and last free segments, adjacent the local log head and local log tail data segments, respectively, within the segment data block. Ongoing identification of the first free and last free segments of a local log device disk, and thus of the log as a whole, is maintained by the log manager routines 74. Thus, the log is operated effectively as a circular data segment buffer spanning the segment data blocks of one or more log device disks. New data segments are written to the log head while log tail data segments are progressively cleaned by the log cleaner routines 76. Meta-data segments are cleaned and, if still valid, relocated to the first free segment log head. User data segments are cleaned and relocated, at least in part, to either the first free segment at the log head or migrated to the main filesystem disks 40. The identifications of the first and last free segments wraps around the log as new data segments are written and old data segments are cleaned from the log as a whole.

Detailed Description Text (116):
Each segment map entry, as individually shown in FIG. 5f, includes a segment map number and a serial number identification. Flag and status information defining the current state of the corresponding data block, and a data block inverse translation completes an individual entry. The inverse translation directly preserves the original disk address of a corresponding data block as stored within a particular data segment. Since both the log device block address of a data segment and the inverse translation for each data block are preserved explicitly on the log device, the data location relationships between data as stored on the log device and within the main filesystem are fully determinable by an ordered evaluation of the most recent set of segment map segments on the log device and any user data segments subsequently and validly written out to the log device.

Detailed Description Text (128):

Where multiple disk drives are used, each of the log disks 122, 124, 126 maintain their own hardware (H/W) and log header blocks specific to their respective disk drive. Each of the drives 122, 124, 126 therefore identifies its own first free data segment and own last free data segment. The ordered relationship between the individual log drives 122, 124, 126 and the first and last free data segment identifiers for the log as a whole is preferably maintained internal to the log manager routines 74. This information is preferably initialized from the superblocks held by each of the log disk 122, 124, 126 by reference to fields within the superblock that serve to describe each log disk as the nth of m log disk for an identified filesystem on the main filesystem disks.

Detailed Description Text (129):
When first initialized, both the head and tail of the active log will exist on the same log disk drive. As data segments are received ultimately from the host computer system through the log device pseudo-device driver, separately illustrated as the write log stream 128, each data segment is stored at the then current first free segment within the log device. Since the physical data layout of the log structured device is entirely hidden from the host, data received from the host is formed into corresponding data segments within the write log stream 128.

Detailed Description Text (130):
The configuration of the log device is, preferably, programmable through IOCTL commands provided through the log device pseudo-device driver, including specifically the write log stream 128. Dynamic configuration and re-configuration through programmable adaptive controls applied to the write log stream 128 on an initialization of the log device pseudo-device driver 61 and, as desired, during the on-going operation of the log device is preferably provided by the user/kernel mode daemons 90. For example, a current default log block size of 8 kbytes may be dynamically reduced to 4 kbyte in order to optimize the operation of the log device to smaller data block writes by the host computer. Other parameters that can be adaptively controlled include the total size of a log segment and the overall size of the log device utilized at one time. Since the write log stream 128, in effect, contains the tables underlying the translation algorithm used to store and retrieve data segments from the log device, changes to the fundamental structure of the physical data layout may be performed dynamically without dependance on existing aspects of either the host computer system or the potentially various filesystem structures present on the main filesystem disks.

Detailed Description Text (131):
The adaptive controls permit fundamental aspects of filesystems provided on the main file system disks to also be dynamically modified. For example, where entire data stripes of a RAID-3 configured filesystem can be held entirely by the log device, the data stripe may be progressively written back to the main filesystem disks in the form of a RAID-5 stripe. This is accomplished by altering the translation algorithm 100 to account for the difference in RAID filesystem layout organization. In effect, many fundamental aspects of a filesystem provided on the main filesystem disks that were previously static can now be dynamically modified through a progressive logging and rewriting of the data to or between filesystems on the main filesystem disks.

Detailed Description Text (132):
In as much as adaptive changes can be applied to the physical layout of the log disks to optimize operation for writing, the present invention thus efficiently permits the filesystem layout on the main filesystem disks to by dynamically altered particularly to optimize operation for data reads.

Detailed Description Text (136):
As part of the data segment cleaning, the relocations information within the user data segment trailer is examined to determine whether any particular log block has been relocated through cleaning in excess of a threshold number of relocations; the

threshold number may be set to an adaptive control defined value. Individual log blocks that have been relocated more than the current relocation value are not incorporated into the new cleaned data segment. Rather, the data blocks within the log block are provided with their corresponding inverse translations to the main disk device driver 34 for writing out to the main filesystem disks 40. In a preferred embodiment of the present invention, the default threshold relocation value is set at three. This value can be modified dynamically to a lower value should the segment data storage space provided by the log device tend to fill at too high a rate. Alternately, the relocation parameter value may be increased to slow the rate of migration of data blocks from the log device to the main filesystem disks. This permits actively or repeatedly written data to be better isolated on the log device for a longer period of time to minimize migration writes to the main filesystem disks and preserve greater bandwidth for main filesystem disk read operations.

Detailed Description Text (138):
However, where a valid match is found, the read log stream 130 is responsible for reading in the corresponding data segment. Since read operations may be random, the requested data segment may lie on any of the log disks 122, 124, 126 within the log device, as illustrated. By utilizing multiple log disks, the chances that the read request must be satisfied from the same log disk that includes the first free segment of the log head is reduced. The operation of the buffer cache within the primary memory 16 further serves, in operation, to reduce the occurrence of disk read requests for a given data block close in time to write data requests for the same data block by the host computer system. Consequently, a substantial majority of read data requests actually satisfied from the log device through the read log stream routines 130 will occur on log disks 124, 126 separate from the log disk 122 that maintains the then current first free data segment of the log. As a result, body and tail log disks 124, 126 can be almost exclusively used for data segment read operations. Conversely, the log disk 122 with the current first free segment of the log is substantially shielded by the operation of the buffer cache within the primary memory 16 and, therefore, performs essentially only write data segment operations.

Detailed Description Text (139):
Thus, a log device system providing for an improved utilization of filesystem read and write operational bandwidth has been described. The log device, appearing to the kernel mode operating system core as the original device driver entry points of the main disk device driver transparently hides both the implementation and operation of a log device subsystem independently implemented on an array of log disks. The resulting effectively composite filesystem established by the combined operation of the log device and the native filesystems supported by the main disk device driver not only allows optimization of read and write data operations, but further allows initially established data layout parameters to be dynamically adjusted to maintain optimal independent response to read and write data requests as the relative nature of such requests change with different application program loads and mixes as executed by the host computer system.

CLAIMS:

1. A method of storing and retrieving data by a computer system executing an operating system and supporting first and second persistent storage devices, said operating system including a filesystem module coupled through a first device driver to said first persistent storage device to transfer filesystem data blocks, said filesystem module providing support for synchronous write transactions, said method comprising the steps of:

a) providing a second device driver selectively coupled between said filesystem module and said first device driver;

b) collecting a predetermined set of data blocks provided from said filesystem module as part of a synchronous write transaction into a data segment, storing said data segment on said second persistent storage device, and signaling completion of said synchronous write transaction to said filesystem module, wherein said step of collecting further provides for

1) constructing a map relating said predetermined set of data blocks to said data segment, whereby said second device driver can identify said data segment by reference to any of said predetermined set of data blocks by said filesystem module; and

2) storing said map, including progressively updated versions of said map, on said second persistent storage device, whereby said map can be reconstructed from said second persistent storage device;

c) migrating said predetermined set of data blocks from said second persistent storage device to said first persistent storage device by said second device driver through use of said first device driver independent of said filesystem module, wherein said data segment includes a predetermined set of address references for said predetermined set of data blocks and wherein said step of migrating said predetermined set of data blocks to said first persistent storage device is performed by said second device driver dependant on said predetermined set of address references;

d) retrieving any of said predetermined set of data blocks from said first persistent storage device;

e) selectively bypass writing said predetermined set of data blocks, as provided from said filesystem module, through said first device driver to said first persistent storage device; and

f) selectively bypass reading said data segment from said second persistent storage device to transfer any of said predetermined set of data blocks to said filesystem module.

3. The method of claim 2 further comprising a step of providing a write data cache within a memory space of said computer system to minimize use of said step of selectively bypass reading.

4. A method of storing and retrieving data within a computer system that is coupleable to a plurality of disk drives, where said computer system executes an operating system, including a filesystem and a main filesystem device driver and where said filesystem defines a data layout for the storage of data blocks within the addressable storage space of a main disk drive, said method comprising the steps of:

a) including a log device driver transparently coupleable between said filesystem and said main filesystem device driver and coupleable to a log disk drive;

b) selectively transferring data blocks between said filesystem, said main disk drive and said log disk drive, wherein data blocks, identified by filesystem address and provided by said filesystem, are preferentially written to said log disk drive and wherein data blocks identified by filesystem address are preferentially read from said main disk drive for transfer to said filesystem;

c) determining, by said log device driver, whether to transfer a predetermined data block, having a predetermined filesystem address, to said main disk drive or said log disk drive based on a data block transfer load comparison between said log device disk drive and said main disk drive, whereby said log device driver seeks to balance the data block transfer load of said log device and main disk drives.

7. The method of claim 6 further comprising the step of writing, to said log disk drive, the main filesystem addresses of said plurality of data blocks in connection with the writing of said log data segment to said log disk drive.

8. The method of claim 7 further comprising the step of relocating said log data segment within the addressable storage space of said log disk drive while maintaining the connection between said predetermined plurality of data blocks and their corresponding main filesystem addresses.

17. A method of storing and retrieving data within a computer system that is coupleable to a plurality of disk drives, where said computer system executes an operating system, including a filesystem and a main filesystem device driver to provide a first addressable storage space, having a first data storage organization defined by said filesystem, within a main disk drive to allow for the storage of filesystem data blocks, said method comprising the steps of:

a) providing a log device driver coupled between and effectively transparent to said filesystem and said main filesystem device driver, said log device driver providing a second addressable storage space within a log device disk drive and having a second data storage organization defined by said log device driver, said second addressable storage space providing for the storage of a predetermined data log segment that includes a predetermined filesystem data block;

b) enabling the selective transfer of said predetermined filesystem data block from said filesystem to said log device disk drive, and from said main disk drive and said log device disk drive to said filesystem; and

c) selecting to transfer said predetermined filesystem data block between said filesystem, said log device disk drive and said main disk drive to preferentially isolating, from the apparent perspective of said filesystem, read transfers of said predetermined data block to said main disk drive and write transfers of said predetermined data block to said log device disk drive, thereby enabling said operating system to transparently access and selectively transfer filesystem data blocks with respect to said second addressable storage space in place of said first addressable storage space.

Previous Doc Next Doc Go to Doc#

Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: L2 and (computer near system)

Display: 50 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Wednesday, November 09, 2005 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L3	L2 and (computer near system)	9	L3
L2	main near filesystem	10	L2
L1	custom near filesystem	3	L1

END OF SEARCH HISTORY

Terms used **custom** and **filesystem** and **computer system** Found **166,357** of **166,357**

Sort results by relevance

Display results expanded form

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search

Try this search in The ACM Guide

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

Best 200 shown Relevance scale

1 Using properties for uniform interaction in the Presto document system

Paul Dourish, W. Keith Edwards, Anthony LaMarca, Michael Salisbury

November 1999 **Proceedings of the 12th annual ACM symposium on User interface software and technology**

Publisher: ACM Press

Full text available: pdf(477.56 KB) Additional Information: full citation, abstract, references, citings, index terms

Most document or information management systems rely on hierarchies to organise documents (e.g. files, email messages or web bookmarks). However, the rigid structures of hierarchical schemes do not mesh well with the more fluid nature of everyday document practices. This paper describes Presto, a prototype system that allows users to organise their documents entirely in terms of the properties those documents hold for users. Properties provide a uniform mechanism for managing, coding, searc ...

Keywords: document interfaces, document management, document properties, interaction models

2 Collecting whole-system reference traces of multiprogrammed and multithreaded workloads

Scott F. Kaplan

January 2004 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 4th international workshop on Software and performance WOSP '04**, Volume 29 Issue 1

Publisher: ACM Press

Full text available: pdf(1.08 MB) Additional Information: full citation, abstract, references, citings

The simulated evaluation of memory management policies relies on *reference traces*--logs of memory operations performed by running processes. No existing approach to reference trace collection is applicable to a complete system, including the kernel and all processes. Specifically, none gather sufficient information for simulating the virtual memory management, the filesystem cache management, and the scheduling of a multiprogrammed, multithreaded workload. Existing trace collectors are al ...

3 Challenges: Challenge:: recombinant computing and the speakeasy approach

W. Keith Edwards, Mark W. Newman, Jana Sedivy, Shahram Izadi

September 2002 **Proceedings of the 8th annual international conference on Mobile computing and networking**

Publisher: ACM Press

Full text available: pdf(297.46 KB) Additional Information: full citation, abstract, references, citings, index terms

Interoperability among a group of devices, applications, and services is typically predicated on those entities having some degree of prior knowledge of each another. In general, they must be written to understand the type of thing with which they will interact, including the details of communication as well as semantic knowledge such as when and how to communicate. This paper presents a case for "recombinant computing" -- a set of common interaction patterns that leverage mobile code to allow r ...

Keywords: mobile code, recombinant computing, serendipitous interoperability, speakeasy

4 Teaching networking and operating systems to information systems majors

D. Robert Adams, Carl Erickson

February 2001 **ACM SIGCSE Bulletin , Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education SIGCSE '01**, Volume 33 Issue 1

Publisher: ACM Press

Full text available: pdf(428.41 KB) Additional Information: full citation, abstract, references, citings, index terms

Teaching networking and operating systems to information systems majors presents many challenges. Operating systems and networking tend to be taught in one of two ways. Either the material is non-technical, directed more toward the *business* information systems major, or the material is overly technical, equivalent of teaching a traditional computer science course. We have developed a model for teaching networking and operating systems to information systems majors that bridges that gap. T ...

5 Article abstracts with full text online: Risks to the public in computers and related systems

Peter G. Neumann

January 2005 **ACM SIGSOFT Software Engineering Notes**, Volume 30 Issue 1

Publisher: ACM Press

Full text available: pdf(211.64 KB) Additional Information: full citation, abstract, index terms

Edited by Peter G. Neumann (Risks Forum Moderator and Chairman of the ACM Committee on Computers and Public Policy), plus personal contributions by others, as indicated. Opinions expressed are individual rather than organizational, and all of the usual disclaimers apply. We address problems relating to software, hardware, people, and other circumstances that affect computer systems. To economize on space, we tersify most items and include pointers to items in the online Risks Forum: (R i j) deno ...

6 Risks to the public: Risks to the public in computers and related systems

Peter G. Neumann

January 2005 **ACM SIGSOFT Software Engineering Notes**, Volume 30 Issue 1

Publisher: ACM Press

Full text available: pdf(211.64 KB) Additional Information: full citation, index terms

7 GASS: a data movement and access service for wide area computing systems

Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, Steven Tuecke

May 1999 **Proceedings of the sixth workshop on I/O in parallel and distributed systems**

Publisher: ACM Press
Full text available: pdf(950.99 KB) Additional Information: full citation, references, citings, index terms

8 A file system for continuous media
David P. Anderson, Yoshitomo Osawa, Ramesh Govindan
November 1992 **ACM Transactions on Computer Systems (TOCS)**, Volume 10 Issue 4
Publisher: ACM Press
Full text available: pdf(1.56 MB) Additional Information: full citation, abstract, references, citings, index terms, review

The Continuous Media File System, CMFS, supports real-time storage and retrieval of continuous media data (digital audio and video) on disk. CMFS clients read or write files in "sessions," each with a guaranteed minimum data rate. Multiple sessions, perhaps with different rates, and non-real-time access can proceed concurrently. CMFS addresses several interrelated design issues; real-time semantics fo sessions, disk layout, an acceptance test for new sessions, and disk schedulin ...

Keywords: disk scheduling, multimedia

9 Running on the bare metal with GeekOS
David Hovemeyer, Jeffrey K. Hollingsworth, Bobby Bhattacharjee
March 2004 **ACM SIGCSE Bulletin , Proceedings of the 35th SIGCSE technical symposium on Computer science education SIGCSE '04**, Volume 36 Issue 1
Publisher: ACM Press
Full text available: pdf(103.18 KB) Additional Information: full citation, abstract, references, citings, index terms

Undergraduate operating systems courses are generally taught using one of two approaches: *abstract* or *concrete*. In the abstract approach, students learn the concepts underlying operating systems theory, and perhaps apply them using user-level threads in a host operating system. In the concrete approach, students apply concepts by working on a real operating system kernel. In the purest manifestation of the concrete approach, students implement operating system projects that run on ...

Keywords: education, emulation, hardware, operating systems

10 Making operating systems more robust: Backtracking intrusions
Samuel T. King, Peter M. Chen
October 2003 **Proceedings of the nineteenth ACM symposium on Operating systems principles**
Publisher: ACM Press
Full text available: pdf(185.10 KB) Additional Information: full citation, abstract, references, citings, index terms

Analyzing intrusions today is an arduous, largely manual task because system administrators lack the information and tools needed to understand easily the sequence of steps that occurred in an attack. The goal of BackTracker is to identify automatically potential sequences of steps that occurred in an intrusion. Starting with a single detection point (e.g., a suspicious file), BackTracker identifies files and processes that could have affected that detection point and displays chains of events i ...

Keywords: computer forensics, information flow, intrusion analysis

11 LIMITS-a system for UNIX resource administration

A. Bettison, F. Adcock, P. Chubb, A. Gollan, C. Maltby

August 1989 **Proceedings of the 1989 ACM/IEEE conference on Supercomputing**

Publisher: ACM Press

Full text available: pdf(1.03 MB) Additional Information: full citation, abstract, references, index terms

The UNIX operating system, despite its emergence as a standard for supercomputer systems, lacks effective support for multiuser resource administration. The design and implementation of a decentralised resource administration system uniformly realisable across the wide variety of UNIX dialects presents a number of problems. Among these problems are potential violations of the UNIX design philosophy, preservation of the user process environment and adherence to industry standards

12 Extending document management systems with user-specific active properties

Paul Dourish, W. Keith Edwards, Anthony LaMarca, John Lamping, Karin Petersen, Michael Salisbury, Douglas B. Terry, James Thornton

April 2000 **ACM Transactions on Information Systems (TOIS)**, Volume 18 Issue 2

Publisher: ACM Press

Full text available: pdf(166.43 KB) Additional Information: full citation, abstract, references, citings, index terms

Document properties are a compelling infrastructure on which to develop document management applications. A property-based approach avoids many of the problems of traditional heierarchical storage mechanisms, reflects document organizations meaningful to user tasks, provides a means to integrate the perspectives of multiple individuals and groups, and does this all within a uniform interaction framework. Document properties can reflect not only categorizations of documents and document use ...

Keywords: active properties, component software, document management systems, document services, user experience

13 Active network vision and reality: lessions from a capsule-based system

David Wetherall

December 1999 **ACM SIGOPS Operating Systems Review , Proceedings of the seventeenth ACM symposium on Operating systems principles SOSP '99**, Volume 33 Issue 5

Publisher: ACM Press

Full text available: pdf(1.87 MB) Additional Information: full citation, abstract, references, citings, index terms

Although active networks have generated much debate in the research community, on the whole there has been little hard evidence to inform this debate. This paper aims to redress the situation by reporting what we have learned by designing, implementing and using the ANTS active network toolkit over the past two years. At this early stage, active networks remain an open research area. However, we believe that we have made substantial progress towards providing a more flexible network layer while ...

14 Disco: running commodity operating systems on scalable multiprocessors

Edouard Bugnion, Scott Devine, Kinshuk Govil, Mendel Rosenblum

November 1997 **ACM Transactions on Computer Systems (TOCS)**, Volume 15 Issue 4

Publisher: ACM Press

Full text available: pdf(400.76 KB) Additional Information: full citation, abstract, references, citings, index terms, review

In this article we examine the problem of extending modern operating systems to run efficiently on large-scale shared-memory multiprocessors without a large implementation effort. Our approach brings back an idea popular in the 1970s: virtual machine monitors.

We use virtual machines to run multiple commodity operating systems on a scalable multiprocessor. This solution addresses many of the challenges facing the system software for these machines. We demonstrate our approach with a prototy ...

Keywords: scalable multiprocessors, virtual machines

15 Novel approaches: High-speed I/O: the operating system as a signalling mechanism

Matthew Burnside, Angelos D. Keromytis

August 2003 **Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications**

Publisher: ACM Press

Full text available: pdf(127.65 KB) Additional Information: full citation, abstract, references, index terms

The design of modern operating systems is based around the concept of memory as a cache for data that flows between applications, storage, and I/O devices. With the increasing disparity between I/O bandwidth and CPU performance, this architecture exposes the processor and memory subsystems as the bottlenecks to system performance. Furthermore, this design does not easily lend itself to exploitation of new capabilities in peripheral devices, such as programmable network cards or special-purpose h ...

Keywords: Architecture, Data Streaming, Operating Systems

16 The evolution of Coda

M. Satyanarayanan

May 2002 **ACM Transactions on Computer Systems (TOCS)**, Volume 20 Issue 2

Publisher: ACM Press

Full text available: pdf(441.35 KB) Additional Information: full citation, abstract, references, citings, index terms

Failure-resilient, scalable, and secure read-write access to shared information by mobile and static users over wireless and wired networks is a fundamental computing challenge. In this article, we describe how the Coda file system has evolved to meet this challenge through the development of mechanisms for server replication, disconnected operation, adaptive use of weak connectivity, isolation-only transactions, translucent caching, and opportunistic exploitation of hardware surrogates. For eac ...

Keywords: Adaptation, Linux, UNIX, Windows, caching, conflict resolution, continuous data access, data staging, disaster recovery, disconnected operation, failure, high availability, hoarding, intermittent networks, isolation-only transactions, low-bandwidth networks, mobile computing, optimistic replica control, server replication, translucent cache management, weakly connected operation

17 The KaffeOS Java runtime system

Godmar Back, Wilson C. Hsieh

July 2005 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 27 Issue 4

Publisher: ACM Press

Full text available: pdf(704.30 KB) Additional Information: full citation, abstract, references, index terms

Single-language runtime systems, in the form of Java virtual machines, are widely deployed platforms for executing untrusted mobile code. These runtimes provide some of the features that operating systems provide: interapplication memory protection and basic system services. They do not, however, provide the ability to isolate applications

from each other. Neither do they provide the ability to limit the resource consumption of applications. Consequently, the performance of current systems degra ...

Keywords: Robustness, garbage collection, isolation, language runtimes, resource management, termination, virtual machines

18 RAID-II: a high-bandwidth network file server

A. L. Drapeau, K. W. Shirriff, J. H. Hartman, E. L. Miller, S. Seshan, R. H. Katz, K. Lutz, D. A. Patterson, E. K. Lee, P. M. Chen, G. A. Gibson

April 1994 **ACM SIGARCH Computer Architecture News , Proceedings of the 21ST annual international symposium on Computer architecture ISCA '94**, Volume 22 Issue 2

Publisher: IEEE Computer Society Press, ACM Press

Full text available: pdf(1.43 MB) Additional Information: full citation, abstract, references, citings, index terms

In 1989, the RAID (Redundant Arrays of Inexpensive Disks) group at U. C. Berkeley built a prototype disk array called RAID-I. The bandwidth delivered to clients by RAID-I was severely limited by the memory system bandwidth of the disk array' s host workstation. We designed our second prototype, RAID-H, to deliver more of the disk array bandwidth to file server clients. A custom-built crossbar memory system called the XBUS board connects the disks directly to the high-speed network, allowing data ...

19 Web-conscious storage management for web proxies

Evangelos P. Markatos, Dionisios N. Pnevmatikatos, Michail D. Flouris, Manolis G. H. Katevenis

December 2002 **IEEE/ACM Transactions on Networking (TON)**, Volume 10 Issue 6

Publisher: IEEE Press

Full text available: pdf(603.11 KB) Additional Information: full citation, abstract, references, index terms

Many proxy servers are limited by their file I/O needs. Even when a proxy is configured with sufficient I/O hardware, the file system software often fails to provide the available bandwidth to the proxy processes. Although specialized file systems may offer a significant improvement and overcome these limitations, we believe that user-level disk management on top of industry-standard file systems can offer similar performance advantages. In this paper, we study the overheads associated with file ...

Keywords: secondary storage, web caching, web performance, web proxies

20 A progress report on SPUR: February 1, 1987

Dave Patterson

March 1987 **ACM SIGARCH Computer Architecture News**, Volume 15 Issue 1

Publisher: ACM Press

Full text available: pdf(408.51 KB) Additional Information: full citation, index terms

L1: Entry 2 of 3 File: USPT May 7, 2002

DOCUMENT-IDENTIFIER: US 6385625 B1
TITLE: Highly available cluster coherent filesystem

Detailed Description Text (9):
In the illustrative embodiment, cluster filesystem 118 includes a cluster filesystem layer 120, a local filesystem 122 and a meta-data stub 124. Generally speaking, a local filesystem is a filesystem designed for computer systems in which only one node accesses a storage device. Accordingly, local filesystems typically do not provide for cluster-wide data coherency. In one embodiment, local filesystem 122 is an off-the-shelf filesystem. Generally speaking, an off-the-shelf filesystem is a non-custom filesystem which enjoys wide distribution and is well supported. Examples of off-the-shelf filesystems are UFS, VxFS, DOS and VMS.

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#
End of Result Set

Generate Collection Print

L1: Entry 3 of 3 File: USPT May 2, 2000

DOCUMENT-IDENTIFIER: US 6058400 A
TITLE: Highly available cluster coherent filesystem

Detailed Description Text (9):
In the illustrative embodiment, cluster filesystem 118 includes a cluster filesystem layer 120, a local filesystem 122 and a meta-data stub 124. Generally speaking, a local filesystem is a filesystem designed for computer systems in which only one node accesses a storage device. Accordingly, local filesystems typically do not provide for cluster-wide data coherency. In one embodiment, local filesystem 122 is an off-the-shelf filesystem. Generally speaking, an off-the-shelf filesystem is a non-custom filesystem which enjoys wide distribution and is well supported. Examples of off-the-shelf filesystems are UFS, VxFS, DOS and VMS.

Previous Doc Next Doc Go to Doc#